ADA - Análisis y Diseño de Algoritmos, 2023-2
Tarea 2: Semanas 5 y 6

y Para entregar el domingo 3 de septiembre de 2023

Problemas conceptuales a las 23:59 por BrightSpace

Problemas prácticos a las 23:59 en la arena de programación

Tanto los ejercicios como los problemas deben ser resueltos, pero únicamente las soluciones de los problemas deben ser entregadas. La intención de los ejercicios es entrenarlo para que domine el material del curso; a pesar de que no debe entregar soluciones a los ejercicios, usted es responsable del material cubierto en ellos.

Instrucciones para la entrega

Para esta tarea y todas las tareas futuras, la entrega de soluciones es *individual*. Por favor escriba claramente su nombre, código de estudiante y sección en cada hoja impresa entregada o en cada archivo de código (a modo de comentario). Adicionalmente, agregue la información de fecha y nombres de compañeros con los que colaboró; igualmente cite cualquier fuente de información que utilizó.

¿Cómo describir un algoritmo?

En algunos ejercicios y problemas se pide "dar un algoritmo" para resolver un problema. Una solución debe tomar la forma de un pequeño ensayo (es decir, un par de párrafos). En particular, una solución debe resumir en un párrafo el problema y cuáles son los resultados de la solución. Además, se deben incluir párrafos con la siguiente información:

- una descripción del algoritmo en castellano y, si es útil, pseudo-código;
- por lo menos un diagrama o ejemplo que muestre cómo funciona el algoritmo;
- una demostración de la corrección del algoritmo; y
- un análisis de la complejidad temporal del algoritmo.

Recuerde que su objetivo es comunicar claramente un algoritmo. Las soluciones algorítmicas correctas y descritas *claramente* recibirán alta calificación; soluciones complejas, obtusas o mal presentadas recibirán baja calificación.

Ejercicios

La siguiente colección de ejercicios, tomados del libro de Cormen et al. es para repasar y afianzar conceptos, pero no deben ser entregados como parte de la tarea.

14.1-2, 14.1-3, 14.2-1, 14.2-4, 14.4-1, 14.4-2, 14.4-5, 14.5-1.

Problemas conceptuales

1. Ejercicio 6.8: *The City of Zion* (Kleinberg & Tardos, página 319).
2. Ejercicio 6.10: *Large computing jobs* (Kleinberg & Tardos, página 321).

Problemas prácticos

Hay cinco problemas prácticos cuyos enunciados aparecen a partir de la siguiente página.

A - Take the Land

Source file name: `land.py`
Time limit: x seconds

The poor man went to the King and said, “Lord, I cannot maintain my family. Please give me some wealth so that I can survive with my wife and children.” The King replied, “I shall grant you a piece of land so that you can cultivate and grow food for your family. In the southern part of the Kingdom there is a rectangular forest. Trees have been planted there at regular intervals. Some of the trees have been cut for use. You are allowed to take any rectangular piece of land that does not contain any tree. You need not go to the forest to select the piece of land. I have a map containing 1’s at places where there is a tree and 0s at points where the tree has been cut.”

Help the poor man to find out the largest piece of land. Area of the land is measured in units of number of trees that were there. Your program should take a matrix of 1’s and 0’s as input and output the area of the largest rectangular piece of land that contain no tree. Be careful about the efficiency of your program.

Input

The input file may contain multiple test cases. The first line of each test case contains two integers M and N ($1 \leq M, N \leq 100$) giving the number of rows and columns in the matrix that follows. Each of the next M lines contains N symbols (either `'0'` or `'1'`). Two consecutive symbols in a line will be separated by a single space. The input terminates with two zeros for M and N.

The input must be read from standard input.

Output

For each test case in the input print a line giving the area (in terms of the number of trees were there) of the largest rectangular piece of land containing no tree.

The output must be written to standard output.

Sample Input	Sample Output

```
6 7
0 1 1 0 1 1 0
0 0 0 0 0 1 0
1 0 0 0 0 0 1
0 1 0 0 0 0 1
1 1 0 0 0 1 0
1 1 0 1 1 0 0
0 0
```
	```
	1220
	0
	220
	```

B - Stock Market

Source file name: `market.py`
Time limit: x seconds

A beginner investor wants to learn how to invest in the stock market. As he does not have any experience, he selected one company and followed daily the value of the stock during N days. At the end, he wondered how much money he would have won if he had invested during the time he followed the stock value. To be honest, the investor is multi billionaire and has a lot of money, enough to buy any amount of stock actions of the company. However, as he is very careful with his investments, he decided that he would never have more than one stock of the company.

To cover his costs, the stockbroker charges a fixed rate of C dollars for every stock purchase.

You have to calculate the maximum profit that the investor could have won investing during the N days, having also the option of not to invest any money.

Input

The input consists of several test cases. The first line of a test case contains two integers, N and C ($1 \le N \le 2 \times 10^5, 0 \le C \le 30$). The second line contains the N prices $P_1, P_2, \ldots, P_N$ of the days $1, 2, \ldots, N$, respectively. Every price P_i satisfies $1 \le P_i \le 1000$.

The input must be read from standard input.

Output

For each test case in the input your program must produce exactly one line, containing exactly one integer, the maximum profit of the investor, in dollars.

The output must be written to standard output.

Sample Input	Sample Output

`6 10`
`100 120 130 80 50 40`
`5 10`
`70 80 50 40 50`
`13 30`
`10 80 20 40 30 50 40 60 50 70 60 10 200`
	`20`
	`0`
	`220`

C - Squares

Source file name: `squares.py`
Time limit: x seconds

For any positive integer N, $N = a_1^2 + a_2^2 + \ldots + a_n^2$ that is, any positive integer can be represented as sum of squares of other numbers.

Your task is to print the smallest n such that $N = a_1^2 + a_2^2 + \ldots + a_n^2$.

Input

The first line of the input will contain an integer t, which indicates the number of test cases to follow. Each test case will contain a single integer N ($1 \leq N \leq 10\,000$) on a line by itself.

The input must be read from standard input.

Output

Print an integer which represents the smallest n such that $N = a_1^2 + a_2^2 + \ldots + a_n^2$.

Explanation for sample test cases:

5 $\rightarrow$ number of test cases

$1 = 1^2$ (1 term)

$2 = 1^2 + 1^2$ (2 terms)

$3 = 1^2 + 1^2 + 1^2$ (3 terms)

$4 = 2^2$ (1 term)

$50 = 5^2 + 5^2$ (2 terms)

The output must be written to standard output.

Sample Input	Sample Output
5 1 2 3 4 50	1 2 3 1 2

D - Stamps and Envelope Size

Source file name: `stamps.py`
Time limit: x seconds

Philatelists have collected stamps since long before postal workers were disgruntled. An excess of stamps may be bad news to a country's postal service, but good news to those that collect the excess stamps. The postal service works to minimize the number of stamps needed to provide seamless postage coverage. To this end you have been asked to write a program to assist the postal service.

Envelope size restricts the number of stamps that can be used on one envelope. For example, if 1 cent and 3 cent stamps are available and an envelope can accommodate 5 stamps, all postage from 1 to 13 cents can be "covered":

Postage	Number of 1 cent Stamps	Number of 3 cent Stamps
1	1	0
2	2	0
3	0	1
4	1	1
5	2	1
6	0	2
7	1	2
8	2	2
9	0	3
10	1	3
11	2	3
12	0	4
13	1	4

Although five 3 cent stamps yields an envelope with 15 cents postage, it is not possible to cover an envelope with 14 cents of stamps using at most five 1 and 3 cent stamps. Since the postal service wants maximal coverage without gaps, the maximal coverage is 13 cents.

Input

The first line of each data set contains the integer S, representing the maximum of stamps that an envelope can accommodate. The second line contains the integer N, representing the number of sets of stamp denominations in the data set. Each of the next N lines contains a set of stamp denominations. The first integer on each line is the number of denominations in the set, followed by a list of stamp denominations, in order from smallest to largest, with each denomination separated from the others by one or more spaces. There will be at most S denominations on each of the N lines. The maximum value of S is 10, the largest stamp denomination is 100, the maximum value of N is 10.

The input is terminated by a data set beginning with zero (S is zero).

The input must be read from standard input.

Output

Output one line for each data set giving the maximal no-gap coverage followed by the stamp denominations that yield that coverage in the following format:

`max coverage =` ⟨ *value* ⟩ `:` ⟨ *denominations* ⟩

If more than one set of denominations in a set yields the same maximal no-gap coverage, the set with the fewest number of denominations should be printed (this saves on stamp printing costs). If two sets with the same number of denominations yield the same maximal no-gap coverage, then the set with the lower maximum stamp denomination should be printed. For example, if five stamps fit on an envelope, then stamp sets of 1, 4, 12, 21 and 1, 5, 12, 28 both yield maximal no-gap coverage of 71 cents. The first set would be printed because both sets have the same number of denominations but the first set's largest denomination (21) is lower than that of the second set (28). If multiple sets in a sequence yield the same maximal no-gap coverage, have the same number of denominations, and have equal largest denominations, then print the set with the lewer second-maximum stamp denomination, and so on.

The output must be written to standard output.

Sample Input	**Sample Output**

```
5
2
4 1 4 12 21
4 1 5 12 28
10
2
5 1 7 16 31 88
5 1 15 52 67 99
6
2
3 1 5 8
4 1 5 7 8
0
```
	```
	max coverage =  71 :  1  4  12  21
	max coverage = 409 :  1  7  16  31 88
	max coverage =  48 :  1  5   7   8
	```

E - Let's Yum Cha

Source file name: `yum-cha.py`
Time limit: x seconds

Yum cha, a term in Cantonese, literally meaning "*drinking tea*', refers to the custom of eating small servings of different foods (*dim sum*) while sipping Chinese tea. It is an integral part of the culinary culture of Guangdong and Hong Kong. For Cantonese people, to yum cha is a tradition on weekend mornings, and whole families gather to chat and eat dim sum and drink Chinese tea. The tea is important, for it helps digest the foods. In the past, people went to a teahouse to yum cha, but dim sum restaurants have been gaining overwhelming popularity recently.

Dim Sum literally means "*touch your heart*", which consists of a wide spectrum of choices, including combinations of meat, seafood, vegetables, as well as desserts and fruit. Dim sum can be cooked, *interalia*, by steaming and deep-frying. The various items are usually served in a small steamer basket or on a small plate. The serving sizes are usually small and normally served as three or four pieces in one dish. Because of the small portions, people can try a wide variety of food.

Some well-known dim sums are:

- **Har gow**: A delicate steamed dumpling with shrimp filling and thin (almost translucent) wheat starch skin. It is one of my favourite dim sum.
- **Siu mai**: A small steamed dumpling with pork inside a thin wheat flour wrapper. It is usually topped off with crab roe and mushroom.
- **Char siu bau**: A bun with Cantonese barbeque-flavoured pork and onions inside. It is probably the most famous dim sum around the world.
- **Sweet cream bun**: A steamed bun with milk custard filling. It is sweet and spongy.
- **Spring roll**: Consists of sliced carrot, wood-ear fungus, and sometimes meat, rolled inside a thin flour skin and deep-fried. It is crispy and delicious.

The picture on the right shows some of the dim sums mentioned above. Can you name them?

Today you go to yum cha with N friends. You and your friends have agreed that everyone will pay the same amount of money (to within one dollar), and each and every one of you will pay at most $\$x$. In the restaurant there are K kinds of dim sums to choose from. Every one of you has assigned an integer "*favour index*" to each dim sum, ranging from 0 to 10.

Now you are responsible for choosing what dim sums to order. You wanted to **maximize your total "*favour value*"** from dim sums you choose, but you will certainly get beaten by all your friends if you ignore their interest. Therefore, you shall **maximize the *mean* of the total favour value everyone gets**, computed using the formula

$$\frac{Total\ favour\ value\ of\ all\ dishes\ ordered}{N+1}$$

instead. Note that even though we are considering the "mean favour value", this does not imply that everyone can actually get a piece of every ordered item!! Anyway, since you are very good friends, you will certainly find some ways to share the food so that everyone is happy, right? :-)

Since you would like to try more different kinds of dim sums, you shall NOT order more than 2 dishes of a same type of dim sum. Also, since you do not want to waste food, you shall NOT order more than $2(N+1)$ dishes in total (i.e. 2 dishes for each of you).

When computing the amount of money to be paid we shall NOT just add up the prices of the dim sums. We also need to take care of the following two charges:

- **Tea charge**: everyone is charged $\$T$ for the tea provided.
- 10% service charge: you need to pay 10% of (Dim sum prices + Tea charge) as service charge. This value is to be rounded up to the nearest dollar.

Constraints:

- $1 \leq N \leq 10$
- $1 \leq x \leq 100$
- $0 \leq T \leq 20$
- $1 \leq K \leq 100$
- $\$1 \leq$ *price of a dim sum* $\leq \$100$.

Input

Input consists of no more than 25 test cases. Each case begins with four integers N, x, T and K, whose meanings have been explained above. Then comes K lines, each giving the information of one particular dim sum. The first integer is the price of the dim sum, followed by your favour index, then N integers which are the favour indices of your N friends.

Input ends with a line with four zeros.

The input must be read from standard input.

Output

For each case, your program should give the optimal mean favour value, correct to 2 decimal places. This value is always positive.

The output must be written to standard output.

Sample Input	Sample Output

`3 10 5 2`
`6 7 5 6 9`
`10 9 10 10 8`
`0 0 0 0`
	`16.00`